OXFORD BASIC

THE COMPLETE BASIC PROGRAMMING ENVIRONMENT *From* S.S.O.L.

commodore 64/64C

OXFORD BASIC

OX-COMP

OX-COMP is the fastest fully compatible BASIC compiler available for the 64. It comes from the manufacturers of OxfordPascal and gives you the speed and professionalism of machine code without the need to write in assembler. OX-COMP is not just a compiler, it also contains a powerful optimiser which breaks down programs into tiny fragments and then re-assembles them simplifying the complex and removing the unnecessary.

Why Use OX-COMP?

OX-COMP will instantly make your BASIC programs run up to 40 times faster.

OX-COMP compiles the whole 64 BASIC language.

OX-COMP makes long programs shorter.

OX-COMP gives you extra features such as long variable names, user defined string functions and fast I/O and more.

OX-COMP lets you mix BASIC and machine code.

OX-COMP allows you to make backup copies of the compiler.

Is OX-COMP easy to use?

Using **OX-COMP** couldn't be simpler. Just type in the name of your program, wait a few minutes and watch your software run up to 40 times faster!

ANALYZER

The OXFORD ANALYZER is a very powerful BASIC programmer's report/optimisation utility designed to give you everything you need to produce high quality software.

ANALYZER can generate the following reports

- ★ Fully sorted GOTO/GOSUB cross reference listings showing clearly the control structure of your program.
- ★ Data cross reference listings showing clearly how each variable, array and function is used. This report is fully sorted and broken down into separate headings for each data type.
- ★ Variable usage frequency report giving useful information concerning the use of each variable.
- ★ A very detailed statistical report showing how much memory is used by all the objects in your program with each variable category and each array broken down separately. The report also includes much additional statistical information.

ANALYZER also features the following program modification/optimisation functions:

- ★ A powerful redundant code eliminator. This function reports all unreachable code and optionally generates a new version of your program with all redundant code removed.
- ★ A Header File Generator. This function dramatically speeds up BASIC and aids documentation by generating a BASIC header program in which all variables are initialized **in the most efficient order.** The TOOLKIT MERGE function may be used to add this code to your program.

COMPRESSOR

The COMPRESSOR is just what its name suggests; a utility for compressing programs down to their minimum size. Just type in the name of your program, wait a minute or two and then see a new version with all spaces and REMs removed and with up to 255 characters packed onto each line.

Why use COMPRESSOR

Because compressed programs contain nothing superfluous, they run much faster and take up less memory and disk space. COMPRESSOR also produces programs which are protected from unauthorised tampering. By using ANALYZER's redundant code elimination function and then running COMPRESSOR on the result, you can produce the smallest possible version of your program. By using ANALYZER's header file generator as well you can produce the fastest possible version of your program that will run under the 64's BASIC interpreter. If you require even more speed, simply compile with OX-COMP!

A complete environment

OXFORD BASIC really is a complete programmers' environment: the combination of TOOLKIT and the report generating features of ANALYZER gives you the means to develop and to automatically document much cleaner and more reliable programs.

The combination of COMPRESSOR and the program modifying features of ANALYZER gives you the means to produce small tight programs of maximum efficiency.

Finally, once your program is complete, you can submit it to OX-COMP and produce an even faster, un-listable piece of highly professional software.

TOOLKIT

Once loaded (by simple menu selection), the OXFORD TOOLKIT fits into an area at the top of RAM and merges itself unobtrusively with the Commodore editor. TOOLKIT does not affect the 64 in any way except to place new commands at your fingertips; commands which will make your BASIC programming more enjoyable and effective.

Here are just a few of the TOOLKIT features:

FIND . . . With the FIND command you can locate ANY string in a BASIC program. Variables, keywords, quoted strings, REMs etc can all be found instantly. You can optionally restrict the search to any given line range and output can be directed to screen, printer or disk.

CHANGE . . . This command works like FIND but also allows you to replace ANY sequence of characters with any other.

DUMP . . . A very useful command; lists all the variables, arrays and functions defined by your program and (in the case of variables) prints out their values. Essential for debugging.

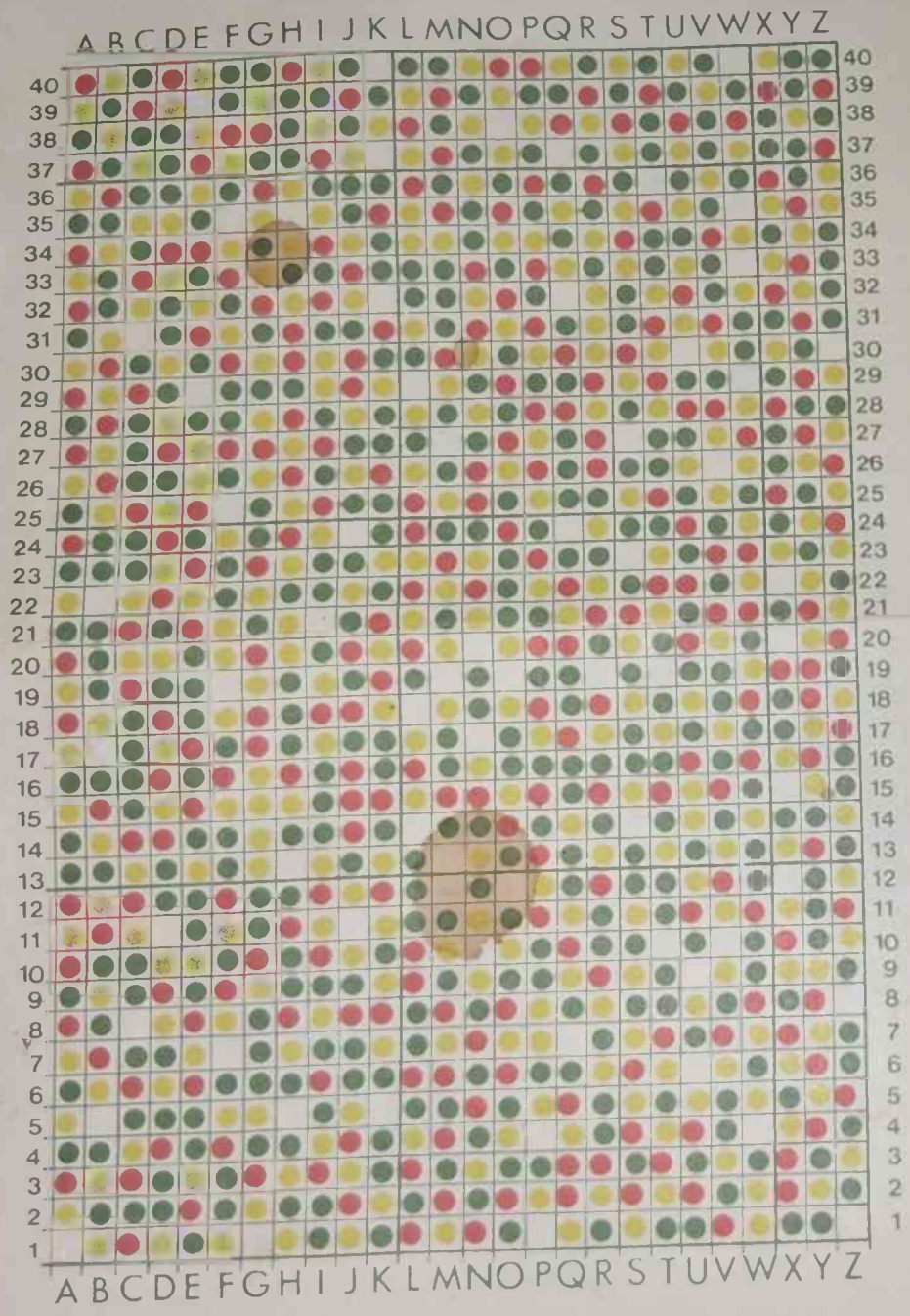
MERGE . . . The MERGE command allows you to join two BASIC programs together into a single file. MERGE lets you take useful routines from existing programs and build them into new code. By using DUMP, CHANGE and MERGE together to ensure against variable clashes, you can build up a library of useful subroutines for inclusion in any program.

INFO . . . This much extended version of the FRE command calculates exactly how much memory you are using for code, variables, arrays and strings and exactly how much memory you have left in each category.

RE-NUMBER . . . A full feature renumber command. Renumber whole programs or just specific routines.

© 1986 Systems Software (Oxford) Limited

16b Worcester Place, Oxford OX1 2JW, England. Tel: 0865·54195 Telex: 83147 OXSOFT



Copyright© 1984 · All rights reserved
SYSTEMS SOFTWARE (OXFORD) LTD
JANUARY 1986

1) LOADING THE SOFTWARE 1

2) THE TOOLKIT 2

Introduction 2

Toolkit Commands 2

3) THE OXFORD BASIC COMPILER 8

Introduction 8

Preparing OX-COMP for Use 8

Running OX-COMP 9

3 Dimensional Arrays 9

Compatibility with BASIC 2.0 10

Facilities not available 10
Extra programming facilities 11

Errors 13

Compile time errors 13
Run time errors 13
The ERRORS utility program 13

Interfacing OX-COMP and Machine code . . . 14

The Memory Map 14
Accessing Variables 14
Variable Formats 15

4) ANALYZER 16

What is ANALYZER 16

Using Analyzer 16

The read-in phase 16
Flow cross reference 17
Data cross reference 18
Dead wood analysis 19
Creating a header file 19
Statistics 20

Displaying Report Files 22

Printing Report Files 22

5) COMPRESSOR 23

What is Compressor 23

Using Compressor 23

MORE PRODUCTS FROM SSOL 24

1: LOADING THE SOFTWARE

Before using OXFORD BASIC it is necessary to follow a brief protection sequence to establish you as an authorised user of the software. We hope that you will not find the protection too tiresome; software does have to be protected and this method at least allows you to make backup copies of the system disk. Proceed as follows:

Mount the OXFORD BASIC System disk, LOAD"OXFORD BASIC",8 and RUN

The screen will now display the following menu

1) Make an OX-COMP/ OX-COMP Utilities master
2) Install Toolkit
3) Run Compressor
4) Run Analyzer
5) Return to BASIC

If you select an option other than 1, OXFORD BASIC will make the protection check at once. If you select option 1 the protection check will not be made until you actually use your OX-COMP master (see the OX-COMP section of this manual).

Oxford BASIC contains so much software that it is supplied on a double sided disk. Note that you may on occasions be requested to remove the Oxford BASIC disk, turn it over and re-insert.

On the sheet provided with OXFORD BASIC will be found a matrix of colored squares. Each square can be identified by means of a simple grid reference. For example to find square E-7, identify column E (marked along the bottom of the matrix) and row 7 (marked along the left hand edge). Square E-7 is where column E and row 7 meet. (see fig 1).

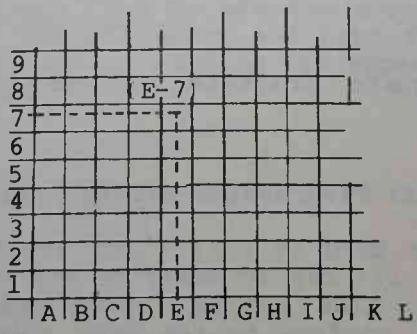


Fig 1.

The protection check is very simple; all you have to do is correctly identify three squares and enter their colors. OXFORD BASIC will give instructions as follows:

WHAT COLOR IS SQUARE X-XX ?

where X-XX is a grid reference. When you have found the square, enter one of the letters W,Y,G or R depending upon whether the square is white, yellow, green or red. When you have correctly answered three such questions the software will be loaded and you can proceed as directed in the documentation relating to the individual programs.

2: TOOLKIT

Introduction

The Commodore 64 provides a very powerful and friendly editor for BASIC programs. Unfortunately, certain features which are normally regarded as essential in a programmers' editor have unaccountably been omitted by CBM. In particular, there is no way on the 64 to find a string in a BASIC program much less to search for an replace one string by another.

The Oxford toolkit attempts to fill this need.

Once loaded, the Oxford toolkit fits unobtrusively into an area at the top of RAM. From this elevated position it provides the programmer with a number of totally new editing features.

The Oxford toolkit does not waste memory by providing nonsense commands that no one ever uses such as BEEP and FLASH! Instead it adds to BASIC ten really useful commands, all designed to improve programming efficiency and productivity.

Loading the Toolkit

To load Toolkit, follow the instructions in the section **loading the Software** at the beginning of this manual.

When Toolkit has loaded it will sign on with the message

```
Oxford TOOLKIT (C) 1986 SSOL.
READY.
```

Toolkit is now fully installed and ready for use.

Toolkit Commands

In the following, square brackets are used to denote optional parameters.

FIND syntax: FIND/search-string/[,line-range]

The FIND command can be used to search for any text string in a BASIC program. The delimiter (/) may actually be any character not contained in the search string. Line ranges, which specify the part of the program to be searched, conform to the same rules as those pertaining to LIST.

Examples

FIND /GOTO100/

will find all occurrences of the string GOTO100 in the currently loaded program

FIND /A=/,1000-2000

will find all occurrences of the string A= in the line range 1000-2000.

FIND !X/2! ,500-

will find all occurrences of the string X/2 from line 500 to the end of the program. Notice that because the search string contains a slash (/),we use shriek (!) as the delimiter.

Tokens and quoted strings

All BASIC keywords such as GOTO, FOR, PRINT and REM are tokenised by the BASIC 7.0 editor, that is, they are held in an encoded form and only decoded into text again by the LIST command. However, if keywords appear in a quoted string, or in a REM or DATA statement, they are left untokenised. When you search for a keyword by typing something like FIND/FOR/, toolkit tokenises the string and then searches for it in tokenised form. FIND/FOR/ would therefore fail to find the FOR in a line like

10 REM PREPARE OUTPUT FOR PRINTING

or

10 PRINT "GONE FOR LUNCH"

To find strings like this we use the special delimiter quote ("). If we use quote as the delimiter and type FIND"FOR", toolkit will find only those lines where the string FOR appears within quotes or in a REM or DATA statement.

Quote always has the effect of switching off tokenisation, even if it appears as part of the search string. For example suppose we wish to find the line

```
10 A$=MID$(B$+"FOR",I)
```

We could type FIND/MID$(B$+"FOR/. Here we don't use quote as a delimiter since if we did, we would fail to find the tokenised MID$. The quote contained in the search string still has the effect of switching off tokenisation for all text following it up until the next quote or delimiter.

CHANGE syntax: CHANGE/search-string/replacement-string/ [,line-range]

CHANGE functions exactly like FIND except that wherever the search string is found, it is replaced by the replacement string. Again, the delimiter may be any character not appearing in the search or replacement strings.

Examples

CHANGE /A$/B$/

will change A$ to B$ wherever it occurs in the program.

CHANGE /GOTO1000/GOSUB1000/,10000-11000

will change any occurrence of the statement GOTO1000 between lines 10000 and 11000 to GOSUB1000

The rules governing quotes are the same as those for FIND. For example, to change the line

```
10 PRINT "GONE FOR LUNCH"
```

to

```
10 PRINT "GONE TO LUNCH"
```

we would type CHANGE"GONE FOR"GONE TO"

DUMP syntax: DUMP [V][A]

DUMP entered without parameters causes the names and values of all variables and the names of all functions and arrays to be displayed.

Examples

Suppose we RUN the following program

```
10 A=1:B=2
20 CC$="OXFORD "+"BASIC":D%=-100
30 DEF FN T(X)=SIN(X)/COS(X)
40 DIM A1(2,3),A2%(A,B)
```

Then the DUMP command will produce the following

```
DUMP
A=1                    B=2
CC$="OXFORD BASIC"     D%=-100
T()                    X=0

A1(2,3)                A2%(1,2)
READY.
```

The V and A parameters which may optionally follow DUMP allow output to be restricted to variables (V) or arrays (A). For instance, using the same example, the command

```
DUMP A
```

would invoke the following output

```
A1(2,3)                    A2(1,2)
READY.
```

INFO syntax: INFO

The INFO command displays some useful information about memory usage in the 64. The information is displayed as follows:

```
         TEXT    VARS    ARRAYS STRINGS
IN USE:  TTTTT   VVVVV   AAAAA  SSSSS
FREE  :  FRE0    .......FRE1..........
```

where

TTTTT is the number of bytes occupied by BASIC text
VVVVV is the number of bytes occupied by simple variables
AAAAA is the number of bytes occupied by arrays
SSSSS is the number of bytes occupied by strings
FRE0 is the number of bytes free excluding variables, arrays and strings
FRE1 is the number of bytes free including variables, arrays and strings

Examples

If we were to run the program illustrated above under DUMP, the INFO command would display the following

```
INFO
          TEXT    VARS    ARRAYS STRINGS
IN USE:   88      42      90      11
FREE  :   54183   .......64113.........
```

MERGE syntax: MERGE"filename",device-no

The MERGE command is very similar to LOAD. The only difference is that whereas LOAD loads a BASIC program and destroys any BASIC currently in memory, MERGE appends the new program to the old one.

Before using MERGE you should ensure that the MERGEd program does not have linenumbers which conflict with program in memory.

Examples

Suppose we have two programs A and B. We wish to merge these into a single program; we proceed as follows

```
LOAD"A",8
LOADING A
READY.
MERGE"B",8
MERGING B
READY.
```

A quick LIST will reveal that both programs are in memory; A followed by B. This may now be SAVEd in the ordinary way.

RENUMBER syntax: RENUMBER [n1] [,n2] [,n3] [-n4]

RENUMBER renumbers a BASIC program with

Parameter	meaning	default
n1	the new starting line number	10
n2	the increment	10
n3	the old starting line number	first line
n4	the old ending line number	last line

Examples

RENUMBER

Renumbers the whole program in steps of 10 starting at line 10

RENUMBER 1000,20,100-199

RENUMBERS the lines 100 to 199 in steps of 20, starting at line 1000

DELETE syntax: DELETE n1-n2

Deletes the lines n1 to n2 inclusive

TYPE syntax: TYPE"filename",device-no

The TYPE command is used to display ASCII files at the terminal. Any file containing printable characters may be TYPEd. When a complete page of the file has been displayed, the prompt ------ MORE ------ appears at the bottom of the screen. Type RETURN to see the next line, SPACE to see the next page or STOP to quit.

SIZE syntax: SIZE"filename",device-no

SIZE is used to determine the size of a file in bytes. It may be used with any file type.

Examples

Suppose we have a sequential file F containing the string "AFFILIATED". To count the number of bytes, we type

```
SIZE"F",8
10 BYTES.
READY.
```

QUIT syntax: QUIT

QUIT de-activates the toolkit.

Reactivating the Toolkit

The toolkit may be reactivated after a QUIT command by typing

SYS 49152.

Memory usage

The toolkit loads at $C000 in the space between kernal and BASIC ROMs. Its presence does not reduce the amount of memory available for BASIC.

3: OX-COMP THE OXFORD BASIC COMPILER

Introduction

OX-COMP is a four pass optimising BASIC compiler compatible with the whole BASIC 2.0 language. OX-COMP generates the fastest running programs of all the BASIC compilers available for the C64.

Preparing OX-COMP for Use

The OXFORD BASIC disk supplied with this manual cannot itself be used to compile programs. Before OX-COMP can be used it is necessary to create two Master disks, a OX-COMP Master and a Utilities Master. When these have been created using the instructions below, they should be backed up and thereafter kept, along with the system disk, in a safe place. When further OX-COMP or utility disks are required, they should be made from the Masters and not the system disk. The system disk should only be used in the event of the Masters becoming corrupt. To create the Masters, proceed as follows:

1) Format 4 disks and mark them OX-COMP MASTER, UTILITIES MASTER, OX-COMP WORK and UTILITIES WORK.

2) Place the OXFORD BASIC disk in the disk drive, LOAD"OXFORD BASIC",8 and RUN. When the menu appears select

   Make an OX-COMP or OX-COMP Utilities master

   as described under Loading The Software at the beginning of this manual. After a short pause the OX-COMP BACKUP menu is displayed. The BACKUP menu provides five options. When one of these is selected the menu will remain on the screen and the selected option will light up in reverse field. When the processing associated with that option terminates, the menu option will be restored to standard video.

3) From the five menu options displayed, select MAKE OX-COMP MASTER and follow the instructions given in the program.

4) When the menu reappears, select MAKE UTILITIES MASTER and again follow the instructions displayed. The system disk may now be put away.

5) Select BACKUP OX-COMP and follow the instructions to make a OX-COMP work disk.

6) Select BACKUP UTILITIES and follow the instructions to make a Utilities work disk.

Compiling a program

Before using OX-COMP it is necessary to follow a brief protection sequence to establish you as an authorised user of the software. This is described in the section Loading The Software at the beginning of this manual.

1 Save the program to be compiled on the OX-COMP WORK disk.

2 LOAD"OX-COMP",8 and RUN.

At this point, OX-COMP makes the protection check.

3 When requested, enter your disk configuration (by menu selection) and the source program name followed by RETURN. The compiler scans the source program four times and as it does so displays the numbers of the lines being processed.

When compilation is complete, the screen will clear and the statement

RUN: NNNNN bytes free

will appear at the top of the screen. The compiled program is now both in memory and saved to disk. To run, simply type the HOME key followed by RETURN.

The compiled program on the work disk has the same name as the original source program but with the characters .WOW appended. It will LOAD and RUN just as a normal BASIC program.

Disk space.

During compilation OX-COMP generates several work files. The largest of these, the parse tree, will often run into several hundred blocks and in some cases there will be insufficient space to accommodate it. Whenever the compiler is used the disk should contain only the OX-COMP system files and the source program.

Three Dimensional Arrays

OX-COMP 64 cannot cope directly with 3D arrays. If a program you wish to compile contains 3D arrays you must use the program PRE-SCAN on your Utilities Work disk. PRE-SCAN takes a BASIC source program as input and produces another BASIC program with all 3D arrays converted into 2D arrays. To use PRE-SCAN proceed as follows:

1 Save the program to be converted onto the Utilities Work disk.

2 LOAD"PRE-SCAN" and RUN.

3 Enter your disk configuration (by menu selection) and then the name of the program to be converted.

PRE-SCAN makes two passes through the program. On completion the disk will contain the converted program. The converted program has the same name as the original program but preceded by the slash "/" character. This program is suitable for compilation.

Compatibility With BASIC 2.0

OX-COMP aims for maximum compatibility with BASIC 2.0. There are some trivial differences but these can be allowed for very easily. They fall into two groups: Facilities not available under OX-COMP and extra facilities provided by OX-COMP.

Facilities not available under OX-COMP

1 When a program is overlayed all variables are normally cleared.

2 OX-COMP needs to know at compile time the dimensions of all arrays. If an array is not dimensioned explicitly in a program it will, as it is in BASIC, be dimensioned by default to eleven elements (0 to 10) per subscript. If an array is dimensioned however it must be dimensioned with constants. Although this slight restriction may sometimes be a problem, it does give OX-COMPed programs the ability to access array elements much more rapidly than would otherwise be possible. Many programs spend a high proportion of their run time accessing array elements and it is a great advantage for a compiler to be able to allocate array space and fix addresses during compilation. For example a statement like

```
100 DIM A(N)
```

is illegal in OX-COMP because the compiler cannot know how much store to reserve. Instead use something like the following

```
100 DIM A(50)
```

If this is not done, the compiler will stop during pass 1 and ask for the dimension as follows:

```
LINE 100: DIM A(
```

Once the dimension is typed in compilation will continue.

3 RUN may be used but not with a linenumber as its parameter. LIST, CONT and SAVE are not allowed.

Because of the overhead of 35 blocks built into all compiled programs, those which occupy less than about 70 blocks of disk space may become larger when compiled. Programs larger than 70 blocks will usually compile smaller. Because OX-COMP saves variables and arrays along with the program, programs will appear larger on the disk than they actually are.

Extra programming facilities

1 User defined string and mixed functions are allowed. Either the function itself or its argument or both may be of either string or numeric type. That is, all the following are possible

```
DEF FN A(X),   DEF FN A$(X),   DEF FN A(X$), DEF FN A$(X$)
```

2 Integer FOR loops such as the following are allowed:

```
FOR I%=0 TO 10: ..... :NEXT I%
```

There is a slight speed advantage in using integer FOR loops. This is because OX-COMP can be sure at compile time that the loop variable will not have a real value assigned to it. The speed advantage is only slight because OX-COMP will try to run all FOR loops as integer loops anyway, only converting to floating point if it is forced to do so.

3 In BASIC 2.0, variable, array and function names can be of any length. The interpreter does however only take notice of the first two characters together with any type descriptor (% or $). To maintain compatibility, OX-COMP treats names in the same way but provides the option of making all characters significant. The following compiler directive is provided for this purpose.

```
REM ! LN
```

This must appear at the beginning of a program before any variables, arrays or functions are defined.

4 Under OX-COMP, the STOP key is disabled by default. Two statements are provided to enable and disable the STOP key when required; these are as follows:

```
REM !ES        (enables the STOP key)
REM !DS        (disables the STOP key)
```

These are statements and NOT compiler directives, that is they will be executed at run time. In the following

```
10 REM !ES
20 GOSUB 100
30 REM !DS
```

the subroutine at line 100 will be executed with the stop key enabled.

The REM !DS statement is to be preferred over other methods of disabling the STOP key in that it has no side effects. Programs also run slightly faster with STOP disabled.

5 The # command.

BASIC 2.0 provides the command CMD in order to redirect the 64's standard output to a file. The use of CMD can often speed up file writing as in the following code fragment which writes 101 strings to disk.

```
10 OPEN 5,8,8,"0:diskfile,s,w"
20 CMD 5
30 FOR I=0 to 100
40 PRINT "testdata"
50 NEXT
60 PRINT#5
20 CLOSE 5
```

Unfortunately BASIC 2.0 provides no equivalent command to redirect the 64's standard input. This is the purpose of the # command. The statement

```
#F
```

where F is the logical file number of an open read file, will cause INPUT or GET to accept input from the file associated with F until the next INPUT#F or GET#F statement is executed. For example, here is a program to read back the data written in the example above:

```
10 OPEN 5,8,8,"0:diskfile,s,r"
20 #5
30 FOR I=0 to 100
40 INPUT A$:PRINT A$
50 NEXT
60 GET#5,a$
20 CLOSE 5
```

ERRORS

All the usual BASIC 2.0 errors CAN occur during compilation and most of these will cause compilation to abort. The only type of compile time error which will not cause the compiler to abort corresponds to BASIC 2.0's UNDEF'D STATEMENT ERROR. Such errors will be found during pass 4 and, in these circumstances, the linenumbers at which the errors occur will appear in reverse field. A program containing references to undefined statements will run, provided that the statements containing such references are never executed. If this should happen the run will terminate with message:

UNDEF'D STATEMENT ERROR IN XXXXX

An error that may occur at compile time where it would not have done in BASIC 2.0 is FORMULA TOO COMPLEX ERROR. This may happen if the 3rd argument of a MID$ contains function or array references. The problem may be easily solved by replacing the 3rd argument with a variable.

Note that occasionally the line number given in an error report is off by one line, that is, the error actually occurred in the line preceding the one reported.

When compilation is aborted due to a compile time error, certain work files are left on the disk. In order to conserve disk space these may have to be scratched prior to the next compilation.

Run Time Errors

With one exception, OX-COMP can produce all appropriate BASIC 2.0 error messages at run time. The exception concerns BASIC's BAD SUBSCRIPT ERROR. Because of the heavy checking overhead this would impose, the OX-COMP run time system does not check that array subscripts are in range. If your program goes wild at run time, it is well worth checking that you have not been writing to non-existent array elements.

All run time errors except UNDEF'D STATEMENT ERROR and BREAK give a code address instead of a linenumber. Make a note of this number. In order to locate the corresponding line, it is necessary to use the error locating program ERRORS on the Utilities Work disk. ERRORS uses a file produced by OX-COMP which has the same name as your program but with the extension .W.

1 Mount the Utilities Work Disk, LOAD"ERRORS",8

2 Mount the petspeed work disk containing .W file for the program that produced the error.

3 Type RUN to start the ERRORS program and select option 2: Locate an error. When requested enter the name of your program and the speedcode address as given in the error report.

After a short delay, ERRORS will respond with the BASIC linenumber where the error occurred.

Interfacing machine code routines to OX-COMP

OX-COMP expects a clean Basic program as a source file. If the program to be compiled contains machine code routines, these should be removed before compilation.

Machine language programs which access BASIC 2.0 variables will not work without alteration as OX-COMP holds its variables and arrays differently. The following notes provide all the information required for making the necessary changes.

Object programs generated by OX-COMP consist of a p-code (Speedcode) optimised version of the source program and a run time interpreter. The full memory map is as follows:

```
------------------------  TOPMEM (pointer at 55/56) --------
DYNAMIC STRINGS
------------------------  TOPCODE (pointer at 2084/2085) ---
CODE
------------------------  STARTCODE (pointer at 2082/2083) -
INITIALISATION DATA
------------------------  STARTINIT (pointer at 2080/2081) -
ARRAYS
------------------------  ARRAYTAB (pointer at 2078/2079) --
DATA STATEMENTS
------------------------  STARTDATA (pointer at 2076/2077) -
VARIABLES
------------------------  10240 ----------------------------
SPEEDCODE INTERPRETER
-------------------------------------------------------------
10 SYS (2073) PETSPEED
------------------------  2048 -----------------------------
PAGE ZERO, STACK, TAPE BUFFERS, SCREEN ETC
------------------------  0 --------------------------------
```

The Speedcode version of a Basic program is usually between 0.5 and 0.65 the size of the original Basic text. To this, however, must be added the 8k overhead of the Speedcode interpreter.

Whereas a BASIC 2.0 program creates its variables and arrays when it is run, a OX-COMP program does so at compile time. TOPCODE is the highest address used by the program except for dynamic strings which are built up from TOPMEM downwards.

Accessing Variables

The location of each variable and array can be found by running the supplied program REPORT on the Utilities disk. The program asks for the source file name and allows output to be directed to screen or printer. REPORT lists all variables, arrays and user defined functions and gives their addresses. In addition, the locations of the main program segments are supplied. The only address missing from the report is that of TOPCODE (see above) which may easily be determined after compilation

The diagram below gives the format of all OX-COMP data types. Corresponding diagrams illustrating the format of BASIC 2.0 variables are to be found in Commodore's own documentation

A word of warning: Variables and array elements which are not declared as integers using the % sign will not necessarily be held in floating point format. The type of the data in an ordinary real variable or array element may change frequently at run time. The way to transfer values from CX-COMP variables to machine code routines is to make them integer explicitly (e.g. A%, PQ%(10,2) etc). If this is not possible, the machine code routine can if necessary obtain the current type of a variable by looking at the type descriptor (see below). Should you wish to pass a value back to CX-COMP via a real/int variable, you may do so by forcing the type descriptor to the value appropriate for the type of data being passed.

Simple Variables (Bytes marked X are not used)

Type	B1	B2	B3	B4	B5	B6	B7
Real/Integer	EXP	M1	M2/MSB	M3/LSB	M4	SGN	0/1
Declared Integer	X	X	MSB	LSB	X	X	5
String	X	X	LENGTH	PTR-LSB	PTR-MSB	X	-ve
Function	X	X	X	PTR-LSB	PTR-MSB	X	X

The current type of Real/Int variables is specified by B7. The 32 most frequently accessed variables are treated specially and contain an extra unused byte (B8).

Arrays

Type	B1	B2	B3	B4	B5
Real/Integer	EXP/0	M1 & SGN/x	M2/MSB	M3/LSB	M4/x
Declared Integer	MSB	LSB	(2 bytes used)		
String	LENGTH	PTR-LSB	PTR-MSB	(3 bytes used)	

The individual elements of Real/Int arrays can also change type. If the exponent is zero, the type is integer, otherwise it is real.

4: ANALYZER

What is Analyzer?

Analyzer is a multi-function program analysis/optimization utility for BASIC programs. Analyzer first reads in a BASIC program and extracts all the information needed to perform its various functions. When the read-in phase is complete, analyzer displays a menu of analysis/optimization options for you to apply.

Preparing to use Analyzer

Prepare a work disk and SAVE onto it all the programs that you wish to analyze. Analyzer will read from this disk and write its own files to it. The work disk should be less than half full.

The read-in phase

To start Analyzer, mount the OXFORD BASIC system disk and proceed as described in Loading the Software at the beginning of this manual. When Analyzer is installed and ready to go you are asked to

MOUNT WORK DISK : HIT RETURN

obey, then enter your

PROGRAM NAME ?

As the read-in phase proceeds, Analyzer will display the numbers of the lines being read. If your program is long this will take a few minutes. When the read-in is complete, Analyzer displays the following menu of options:

1 Flow cross reference

2 Data cross reference

3 Examine disk directory

4 Dead wood analysis

5 Create header file

6 Generate stats

7 Return to BASIC

Options 1,2,4 and 6 allow output to be directed to screen or disk. If you select disk output, all the reports generated by Analyzer are output to sequential files with user specified names. These sequential files may be viewed by using the toolkit TYPE command or printed using the supplied program PRINT.

We now deal with each of the options in turn.

1: Flow cross reference

The flow cross reference consists of a list of target lines, that is, lines referenced by one or more GOTO or GOSUB statements. Along side each target line is a list of all the lines from which it is referenced. For example, the following is a typical flow cross reference output:

SUBROUTINES:-

```
SUB      CALLING LINE(S)

1000 :  10     20     100
2000 :  150    160

GOTO TARGETS

TARGET   REFERENCING LINE(S)

1010 :  1050   1090   1095
2010 :  2090   2095
```

The linenumbers preceded by the colon are the target lines, the linenumbers to the right of the colon are calling lines. Line 2000 for example can be called from lines 150 and 160.

As you can see, the report is split up into SUBROUTINES and GOTO TARGETS. Note that if a given line is referenced anywhere by a GOSUB, the routine at that line will be regarded by Analyzer as a subroutine. All other references to that line will therefore appear under the SUBROUTINES heading regardless of whether the reference is made via a GOTO or GOSUB.

2: Data cross reference

The data cross reference consists of a list of all variables, arrays and functions with each entry followed by a list of the lines where that variable, array or function appears. In addition the function generates a list of all loop variables that is, all variables which are the subject of a FOR loop.

Here is a typical data cross reference output:

```
Functions:

Fl(  :  100   110   200
F2(  :  105   115   200

Arrays:

Al(  :  200   202   500
A2(  :  70    80    505

Integer variables:

A%   :  20    300
PQ%  :  40    50    70    80

String variables

X$   :  40    50    75
SP$  :  40    55    80    90

Real variables

A    :  95    105
I    :  105   115
J    :  100   320   330
J    :  110   200   210

Loop variables:

I     J     K
```

4: Dead wood analysis

This is the smartest thing that Analyzer does. A list is produced of all the lines in your program (if any) that can never be reached when the program is RUN from the beginning. Because of the way BASIC programs are developed, it is common for them to contain a surprising amount of dead wood, much of which the programmer is unaware. Dead wood elimination can be very useful during program development as it minimises memory usage and disk space and reduces loading times. It is particularly useful for producing a final "fair copy" of a finished program with all test routines etc. eliminated.

Note

The dead wood analyzer assumes that you will not wish to CONT after your program has terminated via a STOP or END statement. Code following STOP and END, provided that it cannot be accessed via GOTO or GOSUB, is regarded as un-reachable.

Analyzer displays (or prints) a list of all "dead" lines and then asks

CREATE STRIP FILE (Y/N) ?

If you answer Y, Analyzer copies your program to another file eliminating dead wood as it goes. The new file has the same name as the original but with the characters .s appended.

5: Create header file

This option creates a short BASIC program in which all the variables used in your program are initialized to 0 (or "" in the case of string variables). The variables are initialized in frequency order; that is with the most frequently accessed variables initialized first. The program can be merged with your program using the MERGE function in the LINGO toolkit. This procedure has several merits:

a) Program documentation is improved; Comments can be added to each entry giving details of the purpose of each variable.

b) Execution speed is improved for two reasons

The 64's BASIC interpreter reserves a special area in memory for variables and arrays. Since variables are stored first and arrays last, each time your program uses a variable for the first time, any arrays must be moved up in memory to make room in the variables area. Moving arrays, particularly large ones is time consuming and this explains why BASIC programs often take a little time to get going when they are run. By declaring all variables before the arrays are DIMmed, we can make sure that the interpreter will never need to move the arrays.

When the BASIC interpreter comes across a variable for the first time it simply tags it onto its internal list of variables. When it needs to reference that variable again it has to search through the entire list until it finds the right one. By initializing all variables in an order which reflects their frequency of occurrence, we can usually improve the average search time and thereby improve the speed of the program.

Using the header file generator

When you select option 5 from the Analyzer menu, you will be asked

NUMBER OF STATEMENTS PER LINE ?

enter something sensible. There will then be a short pause after which the header file will be displayed on screen and written to disk under the same name as your program but with a .h appended.

If you request one statement per line, each line of the header file will end with a REM statement. The purpose of this is allow you to document your program by placing along side each variable, a comment describing its use.

When you have finished using Analyzer you can use the MERGE toolkit option to add the header file to the BEGINNING of your program. Needless to say, it is important that the header file should be executed by your program as the first thing it does.

6: Generate Stats

The stats generator provides a breakdown of the various code and data segments in your program and reports the amount of memory occupied by each. Here is a typical stats report

			SEGMENT TOTALS
PROGRAM LENGTH			2594
VAR TYPE	COUNT	TOTAL	
FUNCTION	2	14	
INTEGER	6	42	
STRING	4	28	
REAL	8	56	
TOTAL VARIABLE SIZE			140
ARRAY	NO OF ELEMENTS	SIZE	
A%(4,9)	50	103	
XX(15)	16	80	
TOTAL ARRAY SIZE			183
GRAND TOTAL			2917

LINES	GOTOS	GOSUBS	VARS	ARRAYS
100	7	9	20	2

We will deal with the various entries in order

1) PROGRAM LENGTH - This is the size of the program once it is loaded into the computer. To find the disk file size, add 2 to take account of the load address which occupies the first two bytes of all program files.

2) VARIABLES - VARIABLE TYPE can be FUNCTION, INTEGER, STRING or REAL. Functions are included here because they have a 7 byte entry in the variable table associated with them. COUNT is the number of variables in each category and TOTAL is the total amount of memory occupied in each category (i.e. COUNT*7). All the TOTALs are added together to give a SEGMENT TOTAL for variables.

   The amount of memory required for strings is calculated on the basis that all strings are null (""). As soon as a string is calculated or input (when the program is run) an extra byte is allocated by BASIC for each character in the string. If a string variable is simply assigned to a literal string in the BASIC text, no further space is allocated.

3) ARRAYS - Each array is listed along with its NUMBER OF ELEMENTS and its SIZE inside the computer once it has been DIMmed. An entry is made under SEGMENT TOTAL for the total array storage. In some cases, where an array has been dimmed using an expression and not a constant, Analyzer will be unable to calculate the size. In such cases Analyzer calculates the minimum storage.

4) GRAND TOTAL - The three segment totals are added together to give the total program and data size.

5) The number of lines, variables, gotos gosubs and arrays is reported.

Errors

When analyzing very large programs it is just possible that Analyzer will run out of work space. If this happens, you will receive an error message specifying which type of object is too numerous. Possible error messages are

message	meaning
TOO MANY TARGET LINES	To great a number of different linenumbers have been specified in GOTO or GOSUB statements.
TOO MANY LINE/VAR REFS	The aggregate of all references to linenumbers and variables has exceeded the internal limit.
TOO MANY VARS	The program being analyzed contains too many variables.
TOO MANY LINES	The program being analyzed contains too many lines.

Analyzer allocates space for these objects dynamically and it is therefore not possible to specify an upper limit for the number of any particular type of object. The limits should however be large enough to permit the analysis of any program that will fit into the 64.

If you do get such an error, Analyzer will specify at which line it occurred and then allow reports to be generated for the part of the program successfully analyzed.

Displaying Report Files

When you request a particular report from the Analyzer menu, you will usually be asked whether output is to be directed to the SCREEN or to a DISK file.

If you select SCREEN and the report is too large to fit into a single screen, the first page will be printed and you will thereafter be able to control output by typing P to see the next page, L to see the next line or A to abort the display and return to the menu.

If you select DISK, you will be asked for a

FILENAME?

all output will then be directed to a sequential (SEQ) file of that name. Any existing file of the same name WILL BE DELETED.

If you wish to view a report on screen, you can do so using the TYPE command under the Oxford BASIC Toolkit. This is a useful facility as it allows a report to be displayed while you are editing your program without the necessity to reLOAD.

Printing Report Files

To print a report file, LOAD the program PRINT from the Oxford BASIC disk and RUN. The program will ask for the filename, the number of lines of output required on each page and the number of lines on each page of continuous stationary. If you answer the question

LINES OF OUTPUT PER PAGE?

with a RETURN, PRINT will assume that you want the report printed continuously, without paging. If you answer the question

NUMBER OF LINES ON A PRINTER PAGE?

with a RETURN, PRINT will assume that you are using separate sheets of stationary and will pause after printing each page.
To continue, type C.

5: COMPRESSOR

What is COMPRESSOR

Compressor is a utility for squeezing BASIC programs down into their smallest possible size. First, Compressor removes all "dead wood" such as spaces and REM statements from your program. Next it reduces the number of lines in the program by attempting to pack as many statements onto each line as possible.
Compressor acts on a disk file and writes its output to disk.

Using Compressor

1) Prepare a work disk and save a copy of the program you want to compress onto it. Make sure that there are enough blocks free to accommodate the compressed program.

2) Mount the OXFORD BASIC system disk, LOAD"COMPRESSOR",8 and RUN

After a second or two Compressor will respond with

COMPRESSOR: MOUNT WORK DISK - HIT RETURN

Obey, now you will get the message

PROGRAM NAME ?

Type it in and hit RETURN

Compressor runs through two passes. When it has finished it will display some statistics about your program such as the number of GOTOs and GOSUBs and the total number of lines in both versions of the program.

If you look at the disk directory you will see an extra file with the same name as your original program, but with the characters .c appended. This is the compressed program. LOAD it just as you would any other BASIC program.

See over for details of more SSOL products .....

OTHER PRODUCTS FROM SSOL

All products are available for the C64 and the C128. C128 software is supplied in specially upgraded versions for the 128 using all of this excellent computer's features.

PRODUCT	PRICE		DESCRIPTION
	C64	C128	
Oxford Pascal *	34.95	49.95	A Full, fast Pascal compiler featuring Resident mode, Disk mode, Linker and stand alone Object programs.
Petspeed BASIC compiler	19.95	49.95	The fastest BASIC compiler available!
HACK-PACK (including RAM-DISK)	39.95	39.95	Contains many utilities plus a complete in-RAM "disk drive" which behaves exactly like a much faster 1541/1571.

* Oxford Pascal is also available for the Amstrad CPC6128/PCW8256 and for the BBC micro and Master series.

ORDER FORM (Ref OB64)

Please rush me: _____

for the _____ computer

I enclose £_____

Please make all cheques payable to SSOL

Please include £1.50 p&p and address all orders to

SSOL
16b Worcester Place
Oxford OX1 2JW